

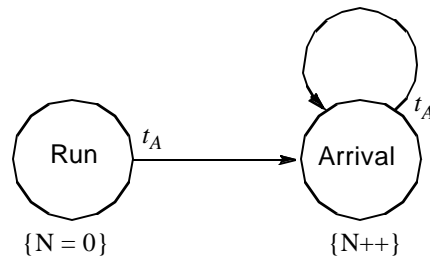
## Computer Lab 1: Introduction to Simkit: The Arrival Process

### Objectives

- Implement simple Event Graph in Simkit
- Learn about `SimEntityBase` class
- Learn methods of `Schedule` to initialize and start simulation
- Generate random variates using `RandomFactory` and `RandomVariate`
- `waitDelay()` method of `SimEntityBase`
- `Schedule.setVerbose()` method
- `Schedule.stopAtTime()` method
- `Schedule.reset()` method
- `Schedule.startSimulation()` method

### The Arrival Process

The discrete-event simulation model of an arrival process specifies a single state, the cumulative number of arrivals, and one event, an arrival, with times between arrivals either deterministic or random. The Event Graph is shown in Figure 1, where  $t_A$  represents successive times between arrivals. Recall that the Run event is the only event in an Event Graph that does not have to be scheduled by another event, but is put on the Event List at time 0.0.



**Figure 1. The Arrival Process Event Graph**

To implement the arrival process using Simkit, perform the following steps:

1. Subclass `SimEntityBase`
2. Define state variables and instance variables
3. Define event methods
4. Write a main method to test

First implement the arrival process with interarrival times constant at 1.1.

Define a subclass of `SimEntityBase` called (imaginatively) `ArrivalProcess`; be sure to put it in your `oa3302` package. The `SimEntityBase` class is in the `simkit` package, so the contents of your file called `ArrivalProcess.java` (in subdirectory `oa3302`) should have (at this point) the following code<sup>1</sup>:

---

1. The code in this handout will not show comments; your program, of course, should be commented according to the OA/OR Programming Standards.

```
package oa3302;
import simkit.*;
public class ArrivalProcess extends SimEntityBase {
}
```

You should now try compiling and running your program. Of course, the program doesn't do anything yet, and without a `main` method, it will not execute at all. Note that you will be responsible for adding comments and features necessary to bring your code into OA3302 Standards compliance. Pop Quiz: what can you conclude about `SimEntityBase`'s constructors given that the above code does indeed compile?

The next step is to define the “event” methods. Simkit is designed so that if you have an Event Graph of your model it is very straightforward to write the code. For starters, every event in the Event Graph corresponds to an instance method in your class that is prefaced with ‘do’.<sup>1</sup> The Event Graph in Figure 1 has two events, Run and Arrival, so you need to define two instance methods `doRun()` and `doArrival()`. They should be `public` and have return type `void` (i.e. no return). Just define the methods with an empty body for now; you will implement them later. Recompile after defining these two methods.

Now define your state variables—in this case, just a single state variable. Each state variable in an Event Graph is implemented as an instance variable in a Simkit model. In order to cut down on clutter, Event Graphs tend to use state variables with short names—`N` for the cumulative number of arrivals in this case. Computer programs, on the other hand, should have variable (and method) names that are as self-documenting as possible. Therefore, your instance variable should be called something like ‘`numberArrivals`.’ (What type do you suppose it should be?) State variables should be declared `protected` and have getter methods, but not setter methods.

You can now put in the state transitions for the `doArrival()` methods; simply read it off the Event Graph in Figure 1. The “state transition” for the Run event occurs in a method called `reset()` in Simkit models:

```
public void reset() {
    numberArrivals = 0;
}
```

The final steps in implementing the Arrival Process are to write the code corresponding to the scheduling edges. Each scheduling edge in a Simkit program is implemented by a call to `waitDelay()` method. There will be two steps (for simplicity). First, make the delays between arrivals have constant value of 1.1. The `waitDelay()` call will have signature `(String, double)`—the first argument is the name of the Event (without the ‘do’), and the second is the delay on that scheduling edge. Your `doArrival()` method should now look like this (Your `doRun()` method should be similar - be careful to implement is as in the Event Graph model shown in Figure 1.):

```
public void doArrival() {
    numberArrivals++;
    waitDelay("Arrival", 1.1);
}
```

At this point you have a complete Simkit model (although it does not yet have all the desired features) so you can put in code to produce some output. You will, of course, need a `main()` method. The `main` method needs to perform several tasks: (1) Instantiate an `ArrivalProcess`; (2) Set the stopping condition; (3) Possibly set other attributes for the simulation run<sup>2</sup>; and (4) Start the simulation.

- 
1. Your class must be a subclass of `SimEntityBase` for this to work.
  2. For this program, the preference is “verbose” mode, in which the current event and the Event List are printed out after every event.

1. *Instantiate an ArrivalProcess*: Define a local variable of type `ArrivalProcess` and ‘new’ it using the zero-parameter constructor.
2. *Set the stopping condition*: Invoke a class method of `Schedule` called `stopAtTime(double)`. The parameter is the simulated time at which the model will stop.
3. *Set verbose mode*: Another static method of `Schedule` is `setVerbose(boolean)`; the parameter tells the model whether or not it is to show the Event List after every event is executed.
4. *Start the simulation*: Invoke the following two methods method of `Schedule`: `reset()` and `startSimulation()`.<sup>1</sup>
5. *Output a report*: After the line invoking `startSimulation()`, write a `println()` statement to output the number of arrivals during the sun.

When you compile and execute your model with a stopping time of 5.0, you should get the following output:

```
** Event List -- Starting Simulation **
0.000   Run
5.000   Stop
  ** End  of Event List -- Starting Simulation **
Time: 0.000   Current Event: Run      [1]
  ** Event List -- **
1.100   Arrival
5.000   Stop
  ** End  of Event List -- **
Time: 1.100   Current Event: Arrival  [1]
  ** Event List -- **
2.200   Arrival
5.000   Stop
  ** End  of Event List -- **
Time: 2.200   Current Event: Arrival  [2]
  ** Event List -- **
3.300   Arrival
5.000   Stop
  ** End  of Event List -- **
Time: 3.300   Current Event: Arrival  [3]
  ** Event List -- **
4.400   Arrival
5.000   Stop
  ** End  of Event List -- **
Time: 4.400   Current Event: Arrival  [4]
  ** Event List -- **
5.000   Stop
5.500   Arrival
  ** End  of Event List -- **
Time: 5.000   Current Event: Stop     [1]
  ** Event List -- **
                << empty >>
  ** End  of Event List -- **
```

- 
1. That is:

```
Schedule.reset();
Schedule.startSimulation();
```

At time 5.0 there have been 4 arrivals

Now that the basic events are working, you need to model random interarrival times. Simkit uses a class called `simkit.random.RandomVariate`<sup>1</sup> that generates random variates with its `generate()` method.

First give your constructor signature (`RandomVariate`), representing the interarrival times for the arrival process. Next, declare an instance variable of type `RandomVariate` called `arrivalTimeGenerator`. Since `arrivalTimeGenerator` is a parameter, it has both a setter and a getter. To use the interarrival times generated, change the second argument in your two `waitDelay()` calls to be:

```
waitDelay("Arrival", arrivalTimeGenerator.generate());
```

Now modify your main method to create and use an instance of `RandomVariate` that are iid exponential(3.2) random variables. Use `RandomVariateFactory` for this as follows:

```
String distribution = "Exponential";
Object[] param = new Object[1];
param[0] = new Double(3.2);
RandomVariate rv = RandomVariateFactory.getInstance(distribution, param);
```

Pass this instance of `RandomVariate` to the constructor of `ArrivalProcess`. Using a stopping time of 15.0, you should get the following output:

```
** Event List -- Starting Simulation **
0.000   Run
15.000   Stop
** End   of Event List -- Starting Simulation **

Time: 0.000      Current Event: Run      [1]
** Event List -- **
0.251   Arrival
15.000   Stop
** End   of Event List -- **

Time: 0.251      Current Event: Arrival  [1]
** Event List -- **
5.769   Arrival
15.000   Stop
** End   of Event List -- **

Time: 5.769      Current Event: Arrival  [2]
** Event List -- **
10.299  Arrival
15.000   Stop
** End   of Event List -- **

Time: 10.299     Current Event: Arrival  [3]
** Event List -- **
10.376  Arrival
15.000   Stop
** End   of Event List -- **

Time: 10.376     Current Event: Arrival  [4]
```

---

1. Actually `RandomVariate` an interface.

```

** Event List -- **
14.857  Arrival
15.000  Stop
** End  of Event List -- **

Time: 14.857      Current Event: Arrival  [5]
** Event List -- **
15.000  Stop
15.371  Arrival
** End  of Event List -- **

Time: 15.000      Current Event: Stop      [1]
** Event List -- **
          << empty >>
** End  of Event List -- **

At time 15.0 there have been 5 arrivals

```

## Finally

Two more additions need to be made to complete the model for this lab. First, the state transition in `doArrival()` should be changed to the following (note the location of `++`):

```
firePropertyChange("arrival", numberArrivals, ++numberArrivals);
```

Then add the following code to your main method after the `ArrivalProcess` is instantiated:

```
SimplePropertyDumper dumper = new SimplePropertyDumper();
arrival.addPropertyChangeListener(dumper);
```

The result should be:

```

** Event List -- Starting Simulation **
0.000  Run
15.000  Stop
** End  of Event List -- Starting Simulation **

Time: 0.000      Current Event: Run      [1]
** Event List -- **
0.251  Arrival
15.000  Stop
** End  of Event List -- **

arrival: 0 => 1
Time: 0.251      Current Event: Arrival  [1]
** Event List -- **
5.769  Arrival
15.000  Stop
** End  of Event List -- **

arrival: 1 => 2
Time: 5.769      Current Event: Arrival  [2]
** Event List -- **
10.299 Arrival
15.000  Stop
** End  of Event List -- **

```

```

arrival: 2 => 3
Time: 10.299    Current Event: Arrival  [3]
  ** Event List -- **
10.376  Arrival
15.000  Stop
  ** End   of Event List -- **

arrival: 3 => 4
Time: 10.376    Current Event: Arrival  [4]
  ** Event List -- **
14.857  Arrival
15.000  Stop
  ** End   of Event List -- **

arrival: 4 => 5
Time: 14.857    Current Event: Arrival  [5]
  ** Event List -- **
15.000  Stop
15.371  Arrival
  ** End   of Event List -- **

Time: 15.000    Current Event: Stop      [1]
  ** Event List -- **
                << empty >>
  ** End   of Event List -- **

```

At time 15.0 there have been 5 arrivals

Note the lines indicating state transitions for the `arrival` state variable that appear just before each Event List snapshot.

## Deliverables

Turn in a hard copies of your program and of your output, which should be identical to the above. Be sure to follow the OA3302 programming standards in your code.

## Frequently Asked Questions

*Why do I need a `doRun()` method?*

Every simulation model needs to be initialized by putting at least one event on the event list at the beginning. Run is the one special event name that is always put on the event list. You create a Run event by writing a `doRun()` method.

*Does every `SimEntityBase` have to have a `doRun()` method?*

No. If a (subclass of) `SimEntityBase` does not define a `doRun()` method then its events must be scheduled by some other `SimEntityBase`. You should generally have at least one `doRun()` method in your model. In next week's lab you will write a class that does not need a `doRun()` method.

*What happens if I don't put a `doRun()` in my model?*

Nothing. Literally. Try it and see.

*How does Simkit know to put the `doRun()` method on the Event List?*

It uses a technique called *reflection*. You can read about reflection in your Java book, but you do not need to understand reflection to write effective models in Simkit.

*Do I have to use `waitDelay()`? Why can't I invoke `doRun()` or `doArrival()` directly?*

You should never directly invoke a ‘do’ method yourself, but should always use `waitDelay()`. The reason is that there is bookkeeping that must be done about which events have occurred and which events are scheduled to occur. Directly invoking a ‘do’ method circumvents this bookkeeping and will cause your model to behave strangely.

*What is the purpose of the `reset()` method?*

The primary purpose of `reset()` is to restore the state variables to their initial values. The current lab does not use `reset()`. However, when you start doing multiple simulation runs, you will need to use `reset()`.

*My program compiles and runs, but after the first Arrival the Event List is empty. What gives?*

You are probably mis-spelling “Arrival” in the `waitDelay()` in `doRun()`. Check to make sure that you have spelled it correctly, including capitalization; if you see ‘arrival’ on the Event list, then you have not capitalized it in your `waitDelay()` method. The same goes for the `waitDelay()` in `doArrival()`.

*Why doesn't `numberArrivals` have a setter method?*

The value of `numberArrivals` should be determined by the simulation model, not arbitrarily set by the program. At any time, the value of `numberArrivals` should be exactly equal to the number of Arrival events that have occurred. In general, state variables should not be given public setter methods, since their values are determined by what has occurred in the simulation.

*Why can't I just use ‘new’ to instantiate the `RandomVariate`?*

Try it and see what happens. As it turns out, you will not be able to instantiate a `RandomVariate` because of the way it is designed. As you will see with future models, the use of `RandomFactory` to get a `RandomVariate` is very flexible and powerful; for now, you'll just have to take my word for it.

*What are those numbers in brackets next to the events on the Event List?*

The numbers in brackets count the number of times each type of event has occurred in the current simulation run.